**HTML/CSS Notes - Week 4**

Date:- 11/03/2022

**Introduction to Sass/Scss**:-

a) Sass (Syntactically awesome style sheets) is an extension of the CSS or a CSS preprocessor, using which we can write CSS in an easy and convenient way.

b) Sass is a Scripting language that is compiled into CSS. SASS can be considered as an extra layer of abstraction added upon CSS to make it easier for the developers to write extensive CSS code.

c) Using SASS/SCSS we can define variables at the top of the styling code, assign it a value, it can be a Hexa-code or anything else and then use the variable in the CSS code. Example:-

| $purple: #6A67CE; $orange: #FF9100;  .purple-btn {  text-align: center;  background-color: $purple; } .orange-text {  color: $orange; } |
| --- |

**Difference between SASS and SCSS**:-

a) There are two syntaxes of Sass i.e SCSS and SASS with two different file extensions ".scss" and ".sass". The sassy CSS or just SCSS is the most commonly used. The syntax of SCSS is similar to the CSS3 syntax.

b) While the SASS is the older version, one of the major differences in writing these two is the use of semicolons and brackets. The older version uses the indentation of the lines to specify the code blocks.

**SASS to CSS(Transpilation)**:- Transpilation is the process using which the SASS/SCSS code is converted into CSS code. It is basically the conversion of one human-readable format into another human-readable format which is SASS/SCSS to CSS.

**Sass Variables**:- Sass uses the $ symbol, followed by a name, to declare variables. Syntax:-

| $variable\_name: value; |
| --- |

Example:-

-> scss file

| $myColor: blue; // global scope by default  h1 {  $myColor: green; // Changing the value of declared CSS variable locally. So it's scope will be limited to this block only.  color: $myColor; }  p {  color: $myColor; } |
| --- |

-> css file(after transpilation)

| h1 {  color: green; }  p {  color: blue; } |
| --- |

-> To change the value of a variable globally, specify '!global' switch along with the name of the variable.

**SCSS Syntax**:-

1. Comments in SCSS- The Scss supports both single-line and multi-line comments i.e // and /\* \*/. But once the Scss or Sass file is compiled into CSS then the resulting CSS will only preserve the multiline comments not the single-line comments, because CSS doesn't support single-line comments.

2. Nested rules- In HTML, we have nested and visual hierarchy. Similar thing can be done while writing SCSS code. Example:-

index.html

| <**html** lang="en">  <**head**>  <**meta** charset="UTF-8" />  <**meta** http-equiv="X-UA-Compatible" content="IE=edge" />  <**meta** name="viewport" content="width=device-width, initial-scale=1.0" />  <**title**>Nested Rules Example</**title**>  <**link** rel="stylesheet" href="./css/style.css" /> </**head**>  <**body**>  <**div** id="sidebar">  <**ul**>  <**li**>Tutorials</**li**>  <**li**>Q & A Forum</**li**>  <**li**>Flashcards</**li**>  <**li**>Tests</**li**>  <**li**>Collaborate</**li**>  </**ul**>  </**div**> </**body**>  </**html**> |
| --- |

style.scss

| #sidebar {  float: left;  width: 25%;  background-color: rgb(119, 137, 143);  padding: 30px 10px;  ul {  list-style-type: none;  margin: 0;  padding: 0;  }  li {  padding: 6px;  margin-bottom: 10px;  background-color: #10A2FF;  color: #ffffff;  } } |
| --- |

3. Parent Selector-In Sass syntax, we can use & symbol to represent the parent class and define further nested rules. This comes in handy when we have to use pseudo-selectors like :hover, :selected etc with the main element. Examples:-

| #demoClass {  // styling for demoClass  &-body {  // styling for #demoClass-body div  } }  a {  text-decoration: none;  color:black;  // using parent selector  &:hover {  color:red;  } } |
| --- |

4. Nested Properties:- In CSS there are many properties that have the same prefix as font-weight, font-size, font-family, etc. In CSS we have to type the complete property name to use them for styling our HTML elements. But Sass syntax allows us to define nested properties with the prefix declared only once. Example:-

| font: {  family: calibri;  size: 12px;  weight: 500; } |
| --- |

**Sass Functions**:- There are many default functions available in Sass, like the darken() function to darken a color, rgb() function to create hexa code for a color etc. Example:-

| $purple: #6A67CE; $darkpurple: darken($purple, 10%);  .purple-btn  {  text-align: center;  background-color: $purple; }  .purple-btn:hover {  background-color: $darkpurple; } |
| --- |

**Sass Operators**:-

The following types of operators are available in SASS.

a) Mathematical or Arithmetic Operators(+, -, \*, /, %):- In SASS, mathematical operators are meant for performing operation on all units of data used in SASS like px, in, etc along with color Hexa codes, so if we perform any mathematical operation on two Hexa code for colors it will produce some other Hexa code. Apart from basic operations, +, - and / operators are also used for String concatenation in SASS.

Example:-

| .maths-operations {  margin: 100px + 200px;  padding: 1000px / 100px - 10px;  height: 5px \* 3px;  background-color: #111100+#001111; } |
| --- |

b) Equality Operators(==, !=)

c) Relational Operators(<, >, <=, >=)

d) String Operators:- There are no special operators in SASS for performing string operations. The mathematical operators +, - and / when used with strings perform the string operations.

-> <expression> + <expression> returns a string that contains both expressions’ values. If either value is quoted-string the result will be quoted; otherwise, it will be unquoted.

-> <expression> / <expression> returns an unquoted string that contains both expression's values, separated by /.

-> <expression> - <expression> returns an unquoted string that contains both expression's values, separated by -. This is a legacy operator but still works.

Example:-

| font-family: "Arial" + " Helvetica" + " sans-serif"; // output will be Arial, Helvetica, sans-serif font-size: 10px / 5px; // output will be 10px/5px and not 2 px font-family: sans-+serif; // output will be sans-serif font-family: sans - serif; // output will be sans-serif |
| --- |

e) Logical Operators:- Logical operators such as 'and', 'or' and 'not' are available in SASS.

**Sass @import and Partials**:-

a) The @import directive allows us to include the content of one file in another. The CSS @import directive has a major drawback due to performance issues; it creates an extra HTTP request each time you call it. However, the Sass @import directive includes the file in the CSS. We do not need to specify a file extension, Sass automatically assumes that we want to include a .sass or .scss file. Example:-

| @import "variables"; @import "colors"; @import "reset"; |
| --- |

b) Sass Partials- When we import a file, we do not need the file to be transpiled or compiled directly. For that to happen, we can start the filename with an underscore. Files named this way are called partials in Sass. Assume we have a file "\_variables.scss" with the content:-

| $myPink: #EE82EE; $myBlue: #4169E1; $myGreen: #8FBC8F; |
| --- |

Now when we import this file into the main scss file, we don't include the underscore

| @import "variables";  body {  font-family: Helvetica, sans-serif;  font-size: 18px;  color: $myBlue; } |
| --- |

**Sass @mixin and @include**:-

a) The @mixin directive lets you create CSS code that is to be reused throughout the website.

Syntax for defining a mixin:-

| @mixin name {  property: value;  property: value; } |
| --- |

Example:-

| @mixin global-styles {  color: lightgray;  font-size: 25px;  font-weight: bold;  border: 1px solid black; } |
| --- |

b) The @include directive is used to include the mixin.

Syntax for defining a mixin:-

| selector {  @include mixin-name; } |
| --- |

Example:-

| .section-content {  @include global-styles;  background-color: green; } |
| --- |

c) A mixin can also include other mixins.

Example:-

| @mixin special-text {  @include global-styles;  @include link;  @include variables; } |
| --- |

d) We can also pass arguments to a mixin. For example:-

| @mixin bordered($color, $width) {  border: $width solid $color; }  .myArticle {  @include bordered(blue, 1px); // Call mixin with two values }  .myNotes {  @include bordered(red, 2px); // Call mixin with two values } |
| --- |

e) Also we can define default values for mixin variables. Example:-

| @mixin border-styles($color: blue, $width: 1px) {  border: $width solid $color; } |
| --- |